Project3

2017030519 컴퓨터소프트웨어학부 홍유진

1. 환경설정 & 실행 방법

- Linux Ubuntu환경에서 진행
- 실행방법 make 후 3_Semantic폴더에서 sematic폴더로 들어간 후 ./cminus sort.cm

```
/3_Semantic/semantic# ./cminus sort.cm
```

2. 과제 내용

(1) 과제 설명

- cminus에서는 global scope이 최상위 scope이다. 따라서 stack에 global scope을 push한 후 compound statement를 만나면 새로운 scope을 스택에 push하고 종료되면 pop한다. 이때 선언되지 않은 변수가 사용되면 error가 발생한다.

- type check

① void는 함수에서만 사용 가능하다.
② Assign할 때 operand들의 type이 같은지 확인한다.
③ return type을 확인한다.
④ Function의 argument member를 확인한다.
⑤ if/while 등의 expression 값을 갖는지 확인한다.

(2) 코드 설명

1) main.c, Makefile

main.c는 과제 pdf를 참고, pdf이미지와는 달리 TraceAnalyze = TRUE로 바꾸어 symbol table을 화면에 나타날 수 있도록한다.
Makefile은 project2와 동일하다.

2) symtab.c , symtab.h

(1) 구조체

```
typedef struct BucketListRec
   { char * name;
     LineList lines;
     TreeNode *treeNode;
     int memloc ; /* memory location for variable */
     struct BucketListRec * next;
   } * BucketList;


typedef struct ScopeRec
   { char * funcName;
     int nestedLevel;
     struct ScopeRec * parent;
     BucketList hashTable[SIZE]; /* the hash table */
   } * Scope;
```

```
typedef struct LineListRec
   { int lineno;
     struct LineListRec * next;
   } * LineList;
```

- ScopeList
 : scope의 이름, bucket의 구조체 배열, 부모의 scope정보를 가지고 있다.
- BucketList
 : call할 때 매개변수와 기존 선언된 함수의 arguments들의 수 타입을 체크할 때 각 트리구조를 미리 집어 넣어놓기 위해 treenode라는 속성을 추가함.
- LineList

(2) 함수

```
static Scope scopes[MAX_SCOPE];
static int nScope = 0;
static Scope scopeStack[MAX_SCOPE];
static int nScopeStack = 0;
static int location[MAX_SCOPE];
```
(symtab.c)

- Scope은 stack으로 관리하기 때문에 새로운 function과 변수들을 symtab.h에 추가하고 symtab.c에 구현해주었다.
- st_insert

```
void st_insert( char * name, int lineno, int loc, TreeNode * treeNode )
{ int h = hash(name);
  Scope top = sc_top();
  BucketList l =  top->hashTable[h];
  while ((l != NULL) && (strcmp(name,l->name) != 0))
    l = l->next;
  if (l == NULL) /* variable not yet in table */
  { l = (BucketList) malloc(sizeof(struct BucketListRec));
    l->name = name;
    l->treeNode = treeNode;
    l->lines = (LineList) malloc(sizeof(struct LineListRec));
    l->lines->lineno = lineno;
    l->memloc = loc;
    l->lines->next = NULL;
    l->next = top->hashTable[h];
    top->hashTable[h] = l; }
  else /* found in table, so just add line number */
  { // ERROR!
  }
} /* st_insert */
```

```
void st_insert( char * name, int lineno, int loc, TreeNode * treeNode );
int st_lookup ( char * name );
int st_add_lineno(char * name, int lineno);
BucketList st_bucket( char * name );
int st_lookup_top (char * name);

Scope sc_create(char *funcName);
Scope sc_top( void );
void sc_pop( void );
void sc_push( Scope scope );
int addLocation( void );
```
(symtab.h)

이런식으로 analyze를 위한 함수를 구현해주었다.

4) analyze.h, analyze.c

1) buildSymtab

: global scope을 scope stack에 push한다. 그리고 빌드 내장함수인 input, output을 global scope의 bucket으로 생성 (내장함수 input, ouput을 만드는 함수는 inoutput()함수에서 구현)

이때 input, ouput을 stack에 push할 때 처음 lineno를 -1로 함께 넣어주어 후에 input oupt을 사용해도 -1은 포함되어 있다.

2) insertNode

: StmtK, ExpK, DecK로 나누어 각각 동작. StmtK는 CompK면 scope을 생성 ExpK는 Id, ArrId, call이 이미 선언되어 있는지 구현, DecK또한 마찬가지로 이미 선언되었는지와 void로 선언되었는지를 확인해준다.

4) typeCheck

: global scope을 스택에 우선 push하고 traverse함수를 통해 type을 check한다.

6) CheckNode

: StmK, ExpK, DecK일 때 각각 다 다르게 행동한다. 그리고 타입에러가 나면 타입에러메시지와 lineno을 출력하게 한다.

전체적으로는 새로운 scope을 만들어 stack에 push하고, state를 빠져나갈 때 stack에서 pop하는 작업을 진행해주었다. parse tree의 node들을 traverser 함수를 이용해 순회하면서 declaration이 있으면 insertNode함수를 이용해 정보를 symbol table에 삽입했다. 만약 자식 노드들이 모두 순회를 마친다면 afterinsertNode 함수로 현재 scope을 stack에서 pop해주었다. symbol table이 완성되면 checkNode를 통해 parse tree를 다시 traverse하여 treenode별로 발생할 수 있는 오류를 각각 맞게 검출해주었다.

3. 실행 결과

1) test.cm

```
TINY COMPILATION: test.cm

Building Symbol Table...

Symbol table:

<global scope> (nested level: 0)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
main           Function  Void          11
input          Function  Integer       -1   14   14
output         Function  Void          -1   15
gcd            Function  Integer        4    7   15

function name: gcd (nested level: 1)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
u              Variable  Integer        4    6    7    7
v              Variable  Integer        4    6    7    7    7

function name: main (nested level: 1)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
x              Variable  Integer       13   14   15
y              Variable  Integer       13   14   15


Checking Types...

Type Checking Finished
```

2)error checking

(1)typeerror.cm

```
TINY COMPILATION: typeerror.cm

Building Symbol Table...

Symbol table:

<global scope> (nested level: 0)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
main           Function  Integer         1
input          Function  Integer        -1
output         Function  Void           -1

function name: main (nested level: 1)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
x              Variable  Integer         3    6
y              Array Var.   4    6


Checking Types...
Type error at line 6: invaild expression

Type Checking Finished
```

(2)undeclare.cm

```
TINY COMPILATION: undeclare.cm

Building Symbol Table...
Symbol error at line 2: undelcared symbol

Symbol table:

<global scope> (nested level: 0)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
main           Function  Integer         1
input          Function  Integer        -1
output         Function  Void           -1

function name: main (nested level: 1)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------


Checking Types...
Type error at line 2: expected return value

Type Checking Finished
```

(symbol error는 symbol table위에 따로출력)

(3)void_var.cm

(4)func.cm

```
TINY COMPILATION: void_var.cm

Building Symbol Table...
Symbol error at line 3: variable should have non-void type

Symbol table:

<global scope> (nested level: 0)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
main           Function  Integer         1
input          Function  Integer        -1
output         Function  Void           -1

function name: main (nested level: 1)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------


Checking Types...

Type Checking Finished
```

```
Symbol table:

<global scope> (nested level: 0)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
main           Function  Integer         6
input          Function  Integer        -1
output         Function  Void           -1
x              Function  Integer         1   12

function name: x (nested level: 1)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
y              Variable  Integer         1    3

function name: main (nested level: 1)
   ID Name      ID Type     Data Type    Line Numbers
-------------  --------  -----------  ------------
a              Variable  Integer         8   12
b              Variable  Integer         9   12
c              Variable  Integer        10   12


Checking Types...
Type error at line 12: the number of parameters is wrong

Type Checking Finished
```